Remote Control Applications using ‘Smart-Controllers’ in Versatile Hardware Configurations

Zack Settel
36 rue de Montmorency
75003 Paris France
email: settel@ircam.fr

Terry Holton
Yamaha Corporation
Professional Audio Division
10-1 Nakazawa-cho
Hammatsu 430, JAPAN

David Zicarelli
Opcode Systems Inc., Suite 100, Palo Alto, CA 94303, USA
zicarelli@well.sf.ca.us

Key words: Remote Control Application, Smart Controller, ISPW, Yamaha DMC1000, Max

Abstract

Each new generation of electronic musical instruments and professional audio equipment is often accompanied by an increase in the “remote controllability” of a given device. As standard communication protocols between machines become faster and more complex, the possibilities for remote control (external control) become more numerous. The MIDI standard, initially used to allow a single musician or computer to play more than one instrument at a time, has become a principal control bus in most studio and live performance applications, both amateur and professional. This standard has been “stretched” to allow devices to control one another in rather sophisticated ways. This paper examines various examples of remote control implementations and strategies, focusing on configurations (i.e. controllers coupled with personal computers (PCs) and devices with external control implementations), that allow users or third-party developers to create applications which take advantage of hardware in ways not necessarily specified by the manufacturers.

1. Examples of Remote Control in Musical and Pro-Audio Applications
1.1 Formatted One-way Communication Between Machines Running Different Programs

A master/slave configuration is often used when one wants to control more than one machine in a synchronized fashion. In many post-production applications, equipment is linked together and controlled by one device, using a control bus that carries control events (start, stop, goto etc.) as well as periodic synchronization signals. MIDI, SMPTE, and MIDI time-code (MTC) are standard protocols that are used in this way. Live performance applications make even greater use of MIDI, allowing a single musician to play or control several sound generators and signal processors (even non-musical devices such as lighting consoles) at the same time. In these applications, communication tends to be one-way: masters dictate to slaves what to execute. Thus protocols tend to be relatively simple, and standardization is easier.

1.2 Formatted Two-way Communication Between Machines Running Different Programs

Other applications exist that require two-way communication between machines. A typical example of this is the “editor-librarian”, which combines a PC with a device (typically a sound generator) whose parameters may be displayed and modified remotely. In such a configuration, the PC acts as a tool that can be used to edit and manage another device’s state. In general, two-way communication is used when it makes sense to maintain a copy of the state of the device on the controlling machine. Messages from the device to the controller generally report the current state of the device (e.g., the “bulk dump” of a set of parameters of a MIDI-controllable sound or effects generator). The controller can act as a database for configurations of the remote device as well as a means of changing its settings.

The HTM system [Freed, 1992] is an interesting implementation that provides for two-way communication between a Macintosh running the Max [Puckette 1990,1991,1991] program and a Silicon Graphics Indigo machine, running HTM. Both machines communicate in real time over the ethernet using UDP protocol [Comer, 1988], thus a musical application may combine the Max control environment with the HTM synthesis environment, and can include several users at one time.

Code downloading often occurs in configurations combining PCs and DSP cards. In these systems, the DSP card depends on the host PC for an interface and file system. These combinations, often based on the IBM PC or the Apple Macintosh, make use of DSP cards incorporating one or more DSP processors (the Motorola 65000 has been very popular). Communication between the host and card takes place over the host’s internal bus. Typically the host will download native code to the card’s DSP chip, tell the DSP to start and stop executing the code, and read and write to addresses on the card which hold parameters of a DSP algorithm. Thus, audio signal processing applications (on the host) may download code (and offload work) to the DSP processor, and communicate with the card at control rates, since the card can handle the audio signal processing and audio I/O portions of the application autonomously. These kind of systems have become quite popular in the consumer and pro-audio markets due to their low cost and the fairly extensive software support offered by third-party developers (Opcode, Stienberg, MOTU etc.).

Another approach has been taken by the manufacturer of a new digital recording processor (Yamaha CBX D5), which is intended to be connected, via SCSI, to a PC and one or more hard disks. Additionally, it is connected to the PC by MIDI. The device handles all DSP tasks such as recording, playback and signal processing, while the PC runs applications (such as sound editors, mixers, cue-lists, sequencers etc.) that control the device and allow users to view and manipulate sound data. An important feature of this configuration is that sound data, stored on hard disk, can be shared between the machines, thus there is no need to copy data from one machine to another. Remote control applications run on the PC, and both machines can read and write sound data, (whose storage file format conforms to that of the PC). Since the device uses standard communication protocols, third party development is easier.

1.3 Two-way Communication Between Different Machines Running the Same Language

The Ircam Signal Processing Workstation (ISPW) [Lindemann, 1990] system is an example of a system that combines a PC (NeXT computer) and a more powerful card for DSP (ISPW board equipped with two general purpose Intel i860 RISC processors). Due to the card’s architecture and power, musical applications written for this platform run on the card, while the host computer provides graphics, events, file I/O, and other support. Thus, in this particular system, communication between host and card (still on the host’s bus) mainly consists of messages passed between different versions of the program (Max language) running simultaneously on each machine. The host, not involved with signal calculation, acts mainly as a user interface for the card, maintaining enough of a copy of the state of the ISPW so that it can represent it on the screen and properly handle user interaction.

The advantage of this approach is that all nongraphical parts of a musical application run on the same processor, in the same language. Furthermore, machine-independent code (Max patches) may be downloaded to the card, which then interprets it. A popular example of machine-independent code downloading is the PostScript language, in which a page of text or graphics is actually a PostScript “program” that tells the imaging device what to draw and how to draw it.

Fundamental parts of a musical application may also run on linked machines that are separated
from one and another, but running the same application. As an experiment, we controlled a sample-playback tool implemented on the ISPW with a Smart-Controller consisting of a Macintosh PowerBook coupled to a MIDI keyboard. Both the ISPW and Macintosh were running Max, and talked to each other by sending and receiving Max messages (formatted into MIDI system exclusive messages). In this configuration, the ISPW served in the manner of a rack-mounted general-purpose DSP device, whose functions are specified by software that is downloaded from, and/or running on the Smart-Controller. The Smart-Controller, on the other hand, serves as a “deluxe” controller with a sophisticated display and I/O (mouse, keys etc.), capable of presenting the user with an appropriate interface and representation of the state of the ISPW, and able to download Max patches to be evaluated and run on the ISPW. Using an remote send (Rsend) object similar to the Max send object, the user can treat the ISPW as if it were an extension of his/her own Max environment, sending and receiving messages to/from any objects in the ISPW Max. This configuration allows us to:

1) Create an application, defined in a single programming language, whose functional parts can be assigned to different platforms. Since Max runs on both the ISPW and the Macintosh, the application’s subpatches are run on the platform which suits the subpatch’s functionality. For example, subpatches handling the graphics (displays, menus etc.), user events (mouse-clicks etc.), and MIDI I/O with the keyboard controller, are run on the Macintosh, while subpatches handling control and DSP are run on the ISPW (our black box).

2) Endow the controller with ‘language-level’ control possibilities. We are free to define arbitrary paths of control between the master and slave. MIDI sound generators are typically limited to notes and note-offs, but one can define the response to any signal on the ISPW in any number of ways. For example, one could define a message “siren” that contained several optional parameters such as base note, oscillation frequency, country of origin, and timbral brightness. When the ISPW received this message from a remote copy of Max, it might initiate a police siren indigenous to the specified country with the indicated base and oscillation frequencies.

3) Model a system in which the Smart-Controller determines the nature of a particular application that makes specialized use of the configuration’s general-purpose hardware. In such a model, the Smart-Controller becomes the task-specific or personalized element of a configuration. It is possible to imagine dedicated systems, ranging from post-production to on-stage performance applications, that differ mainly in terms of Smart-Controllers—and little else.

2. Example of a Remote Control Application

The DMC1000 Project Manager software (PMS), is an application written in Max (and C) for the Yamaha DMC1000 audio mixer. This application runs on the PC, which serves in the manner of a sophisticated remote controller, bi-directionally linked (using MIDI) to the DMC1000. This link between PC and DMC1000 provides a basic real-time communications protocol (using channel and system exclusive messages), allowing the PC to peek and poke values in the DMC’s memory, call functions (without arguments) in the DMC’s system, and receive error, status and acknowledge events from the DMC1000. The PC maintains a copy of the DMC’s state and is able to perform basic operations on the DMC’s data. Thus, the PC may run software (remote control applications) that can complement, extend or enhance the use of the DMC1000 in various areas including: (1) dynamic run-time behavior (eg. additional fader groups), (2) data manipulation (editing, browsing, searching, archiving), (3) user interface (display/edit, control), (4) coordination of linked DMCs in a multiple DMC configuration. The major limiting factors concerning the range of functionality of this application are given by the communication bus’s bandwidth (from CPU, across MIDI, to CPU) and by the degree of machine access defined in the DMC’s system external control implementation. With the current communications bus (MIDI), and the current DMC1000 remote control protocol, we have been able to implement a number of features including: additional fader groups, parameter links, effects editor/librarians, DMC internal disk utilities, graphic display/edit utilities for EQ, and transports for linking automation commands and parameters across multiple DMCs.

3. Future Directions...

Future developments in Smart-Controllers and devices with broad external control implementations should lead to greater separation of control devices from audio processors. This would allow audio processing hardware to become more general-purpose with greater external control; functionality would be determined at run time by remote control applications. Additionally, alternative control interfaces could be produced by third parties, addressing the needs of different users based on their specific preferences or requirements, allowing a greater part of the user’s investment to be spent on features that are specific to their needs.

The future possibilities for remote control would be greatly enhanced if a standard were to be adopted in all electronic devices, from toasters to digital reverberators. Barring this unlikely event, we can suggest an outline of a scheme that would make the flexible control described in the Macintosh-ISPW example more widely available.
Typically, it has been the bias of hardware designers to assign specific codes to specific functions, assuming that computer programmers will rummage through often incoherent "specification documents" in order to create a control application. There has been some attempt within the MIDI industry to standardize the format system exclusive messages understood by all devices. But such a standard is doomed to failure even before it gets off the ground, because it will undoubtedly contain an "escape" for as-yet unforeseen functions invented in new hardware. The presence of several levels of "escape" in the MIDI specification is the very source of its difficulties. Instead of escapes, we need to take advantage of the two-way capability of remote control to allow the controlled device to, in essence, "publish" its system exclusive spec to the controller. The form such a specification would take is trivial: essentially, it would consist of names of operations ("verbs") and parameters and their associated codes and ranges. The controller would receive this information and then construct its user interface or real-time control scheme dynamically. In a more powerful extension of this technique, it would be possible for the master to define new "verbs" for the slave, either in terms of an industry-standard language such as Max or PostScript, or by directly downloading code native to the slave's processor. The master would send a name for the command, an unused code number, a parameter specification, and the machine codes which implement the command. In the simplest case, this would allow for the creation of "macros" which could effectively increase the bandwidth of the communication channel by making a single code stand for a potentially complex combination of operations. In more complex cases, it would endow simple controllers with more "intimate" control possibilities. We are currently experimenting with such a system using two computers running Max. One computer, which plays the role of the master, defines messages that the other, slave, computer implements by downloading a Max patch and then associating the name of one or more messages with particular ways of controlling that patch.

4. Conclusion

It would appear that configurations using Smart-Controllers in conjunction with general-purpose signal processing devices which can be controlled externally, offer some interesting alternatives to the user and developer alike. The user's task-specific investment is largely reduced to Smart-Controllers and software, while third-party developers may develop applications for a greater number of platforms, that perform a wide range of tasks, not necessarily limited by the imaginations of hardware designers. The way one's musical equipment operates is invariably subject to intense modification and scrutiny by the performer, composer, or studio designer; it makes sense to leave most of the details up to motivated individual users or third-party developers. If hardware designers were freed from the burden of attaching musically-unfriendly "interfaces" to their devices, they might gain additional time to pursue the production or modification of sound. At that point, people interested in making flexible devices that specify particular kinds of remote control might find their efforts more rewarded than they have been thus far.

References


Acknowledgements

Hiroshi Hamamatsu, Hirofumi (Hal) Mukaino, Stephan Bilbao.